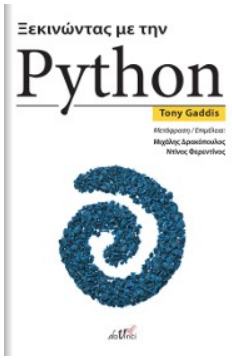
ΠΛΗΡΟΦΟΡΙΚΗ Ι (Python)
Ενότητα 1

Περιεχόμενο μαθήματος:

- Αλγοριθμική επίλυση προβλημάτων
- Προγραμματισμός με Python
- Εφαρμογές σε μαθηματικά και μη προβλήματα

Βιβλίο:



Ανάπτυξη προγραμμάτων: IDLE

Αλγόριθμος: Βήμα προς βήμα διαδικασία για την επίλυση κάποιου προβλήματος. Το πλήθος των βημάτων πρέπει να είναι πεπερασμένο.

- Είσοδος (≥ 0 δεδομένα)
- Έξοδος (≥ 1 αποτελέσματα)
- Ορισμένος (περιέχει σαφείς οδηγίες)
- Κάθε οδηγία, μεμονωμένα: εξαιρετικά απλή
- Καλύπτει όλες τις δυνατές καταστάσεις
- Εξασφαλίζει τον τερματισμό (πεπερασμένος αριθμός βημάτων ή χρόνος)

Πρόγραμμα: Ακριβής διατύπωση ενός αλγορίθμου σε μια γλώσσα προγραμματισμού.

Τρόποι αναπαράστασης/περιγραφής αλγορίθμων:

- Φυσική γλώσσα
- Διάγραμμα ροής (flowchart)
- Ψευδοκώδικας (pseudocode)
- Κώδικας

Υπολογιστική επίλυση προβλήματος:

1) Ανάλυση δεδομένων του προβλήματος
2) Μαθηματική διατύπωση του προβλήματος
3) Ανάπτυξη του ***αλγόριθμου*** (σχεδιασμός ή επιλογή κατάλληλου αλγόριθμου
   (συνήθως: ένα πρόβλημα – πολλοί αλγόριθμοι)
4) Διατύπωση αλγόριθμου σε γλώσσα προγραμματισμού: ***πρόγραμμα***
5) Εκτέλεση προγράμματος για συγκεκριμένα δεδομένα
6) Ερμηνεία αποτελεσμάτων

Βασικές αλγοριθμικές ενέργειες

- Είσοδος / Έξοδος
- Πράξεις ή αναθέσεις τιμών σε μεταβλητές
- Έλεγχος ποσοτήτων – επιλογή δράσης
- Επαναληπτική εκτέλεση
- Τερματισμός

Παράδειγμα αλγόριθμου – Ταξινόμηση φυσαλίδας (bubble sort)

Πρόβλημα: Να ταξινομηθεί κατά αύξουσα σειρά μία λίστα αριθμών.

*Ο αλγόριθμος της φυσαλίδας σε **φυσική γλώσσα**:*

1) Σύγκρινε κάθε ζευγάρι στοιχείων (1ο με 2ο, 2ο με 3ο, ..., (ν-1)ο με ν-οστό) και εάν τα στοιχεία κάποιου ζευγαριού είναι λάθος ταξινομημένα, ενάλλαξέ τα (swap)
2) Εάν έγινε έστω και μία εναλλαγή στοιχείων στο προηγούμενο βήμα, τότε επανέλαβε τη διαδικασία σύγκρισης των ζευγαριών (βήμα 1), διαφορετικά τερμάτισε τον αλγόριθμο.
   (σε κάθε νέα επανάληψη του 1ου βήματος, το τελευταίο ζευγάρι της προηγούμενης επανάληψης της διαδικασίας δε χρειάζεται να συγκριθεί, άρα συγκρίνεται πάντα ένα ζευγάρι λιγότερο)

Π.χ.,

Αρχική λίστα:	5	3	8	1	4	5	7	6
1η σύγκριση:	3	5	8	1	4	5	7	6
2η σύγκριση:	3	5	8	1	4	5	7	6
...								
(*n*-1)η σύγκριση:	3	5	1	4	5	7	6	8
Μετά το 1ο πέρασμα:	3	5	1	4	5	7	6	8
...								
Μετά το 2ο πέρασμα:	3	1	4	5	5	6	7	8
...								

Γλώσσες προγραμματισμού

i. Γλώσσες υψηλού επιπέδου (Γ.Υ.Ε.) (Java, C, C++, Python, Fortran, Pascal, κ.α.)
ii. Γλώσσες χαμηλού επιπέδου (γλώσσα μηχανής (Γ.Μ.), assembly)

Οι Η/Υ εκτελούν γλώσσα μηχανής

Γ.Μ.:

- διαφορετική για κάθε τύπο επεξεργαστή
- εντολές = αλληλουχίες από bits (0 και 1)
- οι εντολές εκτελούνται άμεσα από τη CPU

Assembly (λίγο ανώτερο επίπεδο από Γ.Μ.):

- διαφορετική για κάθε τύπο επεξεργαστή
- μνημονικά ονόματα αντί για bits (ίδιο πλήθος εντολών)
- μετατροπή σε Γ.Μ. από κατάλληλο πρόγραμμα (assembler)

Γ.Υ.Ε.:

- Ανεξάρτητες από τον τύπο επεξεργαστή (υψηλή φορητότητα)
- Υψηλή αφαίρεση / ευκολότερος προγραμματισμός
- Μετατροπή σε Γ.Μ. από κατάλληλα προγράμματα:
  i) μεταγλωττιστές (compilers) ή ii) διερμηνείς (interpreters)

Μεταγλωττιστές (Compilers)

- Προγράμματα που *μετατρέπουν ένα πηγαίο πρόγραμμα (Γ.Υ.Ε.) σε εκτελέσιμο (Γ.Μ.)*.
- Το παραγόμενο εκτελέσιμο πρόγραμμα "τρέχει" αυτόνομα σε κάθε υπολογιστή που "καταλαβαίνει" τη συγκεκριμένη Γ.Μ. (δεν απαιτείται πλέον ο μεταγλωττιστής).

Διερμηνείς (Interpreters)

- Προγράμματα που *εκτελούν άμεσα* ένα πηγαίο πρόγραμμα (Γ.Υ.Ε.).
- Δεν παράγουν ολοκληρωμένο εκτελέσιμο κώδικα, αλλά *ανά εντολή*.
- Για την εκτέλεση του πηγαίου προγράμματος σε κάποιον υπολογιστή *απαιτείται και η ύπαρξη του διερμηνέα*.
- Τα ερμηνευόμενα προγράμματα είναι *πιο αργά* από τα μεταγλωττισμένα.

Σφάλματα προγραμματισμού

α) Συντακτικά σφάλματα (syntax errors)
- Ανιχνεύονται από τον compiler / interpreter

β) Σφάλματα κατά την εκτέλεση (run-time error)
- Ανιχνεύονται από τον interpreter (σε γλώσσες που διαθέτουν)

γ) Λογικά σφάλματα (logic errors / bugs)
- Ανιχνεύονται από τον προγραμματιστή (το πρόγραμμα εκτελείται αλλά το αποτέλεσμα είναι λάθος)

- Δυσκολία εντοπισμού σφαλμάτων (συνήθως): **γ > β > α**
- Διαδικασία εντοπισμού και επίλυσης σφαλμάτων: **debugging**

Βασικά χαρακτηριστικά της Python

- Γλώσσα προγραμματισμού υψηλού επιπέδου
- Υποστηρίζει τόσο Διαδικαστικό όσο και Αντικειμενοστραφή Προγραμματισμό
- Χρησιμοποιεί διερμηνέα και είναι scripting language
- Εκτελείται:
  - είτε ανά εντολή (εντολή = μία ή πολλές γραμμές κώδικα) στο Python shell / κέλυφος (Διαδραστική λειτουργία)
  - είτε ως πλήρες αρχείο προγράμματος με κατάληξη .py (Λειτουργία script)
- Δυναμική γλώσσα στον ορισμό τύπου μεταβλητών
- Εύκολη στη σύνταξη και στην εκμάθηση, κατανόηση και ανάγνωση
- Υποχρεωτική στοίχιση (indentation), όχι μόνο για μορφοποίηση
- Δημιουργία μικρότερων σε μέγεθος προγραμμάτων σε σχέση με άλλες γλώσσες προγρ/σμού
- Μεγάλη πρότυπη βιβλιοθήκη (standard library) έτοιμων συναρτήσεων
- Πληθώρα υλοποιήσεων: Cpython, Cython, Jython, PyPy, ...
- Πληθώρα IDEs: IDLE, Ipython, PythonAnywhere (on-line), ...

Ιστορικά στοιχεία

- Αναπτύχθηκε από τον Ολλανδό Guido van Rossum στις αρχές της δεκαετίας του '90
- Πρώτη έκδοση: Python 0.9.0 – Φεβρουάριος 1991
- Python 1.0 – Ιανουάριος 1994
- Python 2.0 – Οκτώβριος 2000
- Python 3.0 (backwards-incompatible) – Δεκέμβριος 2008
- Τρέχουσα (Οκτώβρης 2015) έκδοση: Python 3.5.0 (13 Σεπτεμβρίου 2015)

Τρόποι υλοποίησης ενός προγράμματος Python

1. *Διαδραστική Λειτουργία*

Εκκινώντας το Python IDE (IDLE) ή γράφοντας την εντολή `python` στη γραμμή εντολών του λειτουργικού συστήματος, εμφανίζεται κάτι σαν το παρακάτω:

```
Python 3.4.3 (default, Mar 26 2015, 22:03:40)
[GCC 4.9.2] on linux
Type "copyright", "credits" or "license()" for more information.
>>>
```

Με την πληκτρολόγηση κάποιας εντολής μετά το `>>>` και το πάτημα του ENTER, αυτή διερμηνεύεται και εκτελείται.

2. *Λειτουργία Script*

   i. Ο κώδικας του προγράμματος συντάσσεται σε κάποιον κειμενογράφο (editor) και αποθηκεύεται σε αρχείο με κατάληξη `.py` (π.χ., `hello.py`)

ii. Διερμηνεύεται και εκτελείται η κάθε εντολή του προγράμματος, ως εξής:
   - Με την εντολή `python hello.py` στη γραμμή εντολών, ή
   - Με τη χρήση του *Run* στο παράθυρο του Editor του IDLE

Προγραμματιστικά Σχόλια

Το σύμβολο # χρησιμοποιείται για την είσοδο προγραμματιστικών σχολίων σε ένα πρόγραμμα. Οτιδήποτε βρίσκεται στα δεξιά του συμβόλου # δεν λαμβάνεται υπόψη από τον διερμηνέα της Python.

Έξοδος (Output)

Με χρήση της συνάρτησης **`print`**: Εκτύπωση στην οθόνη

Σύνταξη:

```
print("συμβολοσειρά εξόδου")
```

ή

```
print('συμβολοσειρά εξόδου')
```

Παράδειγμα στη διαδραστική λειτουργία:

```
>>> print('Hello world')
Hello world
>>>
```

Παράδειγμα στη λειτουργία script:

i) Δημιουργία νέου αρχείου (File -> New File) και σύνταξη προγράμματος:

```
print('Hello world')
```

ii) Αποθήκευση του αρχείου, π.χ. `hello.py`

iii) Εκτέλεση του αρχείου: α) Μέσω της επιλογής *Run* -> *Run Module* του editor (ή F5), ή
β) Μέσω της γραμμής εντολών: `python hello.py`

iv) Έξοδος: `Hello world`

Βασικά στοιχεία της `print`:

- Όταν η συμβολοσειρά εξόδου περιέχει απόστροφο (`'`), μπορούν να χρησιμοποιηθούν τα διπλά εισαγωγικά στην `print` για την οριοθέτηση της συμβολοσειράς:

```
print("Ό,τι να'ναι")
```

- Όταν η συμβολοσειρά εξόδου περιέχει διπλά εισαγωγικά (`"`), μπορούν να χρησιμοποιηθούν τα μονά εισαγωγικά στην `print` για την οριοθέτηση της συμβολοσειράς:

```
print('Η ταμπέλα γράφει "Προσοχή! Ο σκύλος δαγκώνει".')
```

- Όταν η συμβολοσειρά εξόδου περιέχει και απόστροφο και διπλά εισαγωγικά, χρησιμοποιούνται τριπλά εισαγωγικά (`"""` ή `'''`) για την οριοθέτηση της συμβολοσειράς:

```
print("""Διαβάστε τον "Άμλετ" απ'την αρχή.""")
```

- Τα τριπλά εισαγωγικά μπορούν να χρησιμοποιηθούν και για οριοθετήσουν συμβολοσειρές πολλαπλών γραμμών:

```
print('''Ένα
Δύο
Τρία''')
```

Η εντολή αυτή θα εμφανίσει:

```
Ένα
Δύο
Τρία
```

(Περισσότερα για την `print` στη συνέχεια...)

Μεταβλητές (Variables)

- Ονόματα που αναπαριστούν τιμές αποθηκευμένες στη μνήμη του υπολογιστή (αναφέρονται σε αυτές).
- Χρησιμοποιούνται για την αποθήκευση και επεξεργασία δεδομένων.
- Το στοιχείο που περιέχει μια μεταβλητή: **τιμή**

Μια μεταβλητή έχει:

- τύπο (type)
- όνομα (identifier – αναγνωριστικό)
- τιμή (value)

Δημιουργία μεταβλητής: Με εντολή εκχώρησης

`μεταβλητή = τιμή`
ή
`μεταβλητή = έκφραση`

Ο τελεστής = ονομάζεται ***τελεστής εκχώρησης***: εκχωρεί την τιμή που βρίσκεται δεξιά του στη μεταβλητή που βρίσκεται αριστερά του. (Γι αυτό και η εντολή `τιμή = μεταβλητή` είναι *λάθος*)

Π.χ.,

```
>>> age = 25
>>> print(age)
25
>>>
```

Προφανώς, όταν το όρισμα της συνάρτησης `print` είναι μια μεταβλητή, αυτό που εκτυπώνεται είναι η τιμή της και όχι η συμβολοσειρά του ονόματός της.

➔ Στην Python δε χρειάζεται να ορισθεί ο τύπος της μεταβλητής, όπως γίνεται στις περισσότερες γλώσσες προγραμματισμού. Ο τύπος της προσδιορίζεται αυτόματα από την Python (και μπορεί να αλλάξει, όπως θα δούμε αργότερα).

Η τιμή μιας μεταβλητής μπορεί να αλλάξει, π.χ.

```
>>> age = 25
>>> print(age)
25
>>> age = 22
>>> print(age)
22
```

Όταν μια τιμή στη μνήμη δεν αναφέρεται πλέον από κάποια μεταβλητή (όπως στο παραπάνω παράδειγμα το 25), ο διερμηνέας της Python την αφαιρεί αυτόματα από τη μνήμη (garbage collection).

Βασικοί κανόνες ονοματολογίας

- Δε μπορούν να χρησιμοποιηθούν δεσμευμένες λέξεις (λέξεις-κλειδιά): class, for, if, else, while, κτλ.
- Ξεκινάνε με: γράμμα ή κάτω παύλα (_)
- Περιέχουν: γράμματα, αριθμούς, _ (όχι τελείες, *, # κτλ.)
- χωρίς κενά
- υπάρχει διάκριση μεταξύ κεφαλαίων και μικρών (case sensitive)

Συνήθως (κατά σύμβαση):
- σταθερές → όλα ΚΕΦΑΛΑΙΑ
- περιγραφικά ονόματα (εκτός από τις μεταβλητές μετρητών, που προτιμούνται σύντομα ονόματα)

- ξεχωρίζουμε λέξεις ενός ονόματος συνήθως με _
  π.χ., `number_of_baskets`

Π.χ. κάποια ονόματα μεταβλητών:

`my.var`: λάθος (περιέχει τελεία) `7eleven`: λάθος (ξεκινάει με αριθμό)
`else`: λάθος (λέξη-κλειδί) `arithmos_foititwn`: σωστό

Τύποι μεταβλητών

α) Αριθμητικοί τύποι:
i) Τύπος **`int`** για ακέραιους αριθμούς
ii) Τύπος **`float`** για πραγματικούς αριθμούς

β) Συμβολοσειρές (strings): Τύπος **`str`**

Όταν δημιουργείται μία μεταβλητή με κάποια εντολή εκχώρησης, ο διερμηνέας της Python προσδιορίζει αυτόματα τον τύπο της, ανάλογα με την τιμή της. Αν η τιμή της είναι αριθμός χωρίς υποδιαστολή (.) τότε προσδιορίζεται ως τύπου `int`, αν είναι αριθμός με υποδιαστολή προσδιορίζεται ως τύπου `float`, και αν η τιμή της είναι συμβολοσειρά, προσδιορίζεται ως τύπου `str`. Μπορεί κάποιος να ελέγξει τον τύπο κάποιας μεταβλητής με τη χρήση της ενσωματωμένης συνάρτησης `type` σε διαδραστική λειτουργία, όπως στο παρακάτω παράδειγμα δημιουργίας τριών μεταβλητών:

```
>>> room = 506
>>> price = 3.99
>>> name = 'George'
>>> type(room)
<class 'int'>
>>> type(price)
<class 'float'>
>>> type(name)
<class 'str'>
```

Μέσα σε πρόγραμμα, η `type` πρέπει να χρησιμοποιείται μέσα σε συνάρτηση `print`, η οποία εκτυπώνει το κείμενο το οποίο επιστρέφει η `type` όταν εκτελείται: `print(type(room))`

Είσοδος (Input)

Με χρήση της συνάρτησης **`input`**: Ανάγνωση από το πληκτρολόγιο

Σύνταξη:

```
μεταβλητή = input('συμβολοσειρά μηνύματος προτροπής')
```

Π.χ.,

```
name = input('Πώς σε λένε;')
```

Σημαντικό:

Η συνάρτηση `input` επιστρέφει συμβολοσειρά, δηλαδή στη μεταβλητή που βρίσκεται πριν τον τελεστή εκχώρησης = εκχωρείται πάντα συμβολοσειρά (ακόμα και αν ο χρήστης πληκτρολογήσει κάποιον αριθμό). Άρα, στο παρακάτω παράδειγμα:

```
age = input('Δώσε την ηλικία σου: ')
```

εάν ο χρήστης πληκτρολογήσει έναν ακέραιο, π.χ., 20, τότε η μεταβλητή `age` "περιέχει" τη συμβολοσειρά `'20'` και όχι τον ακέραιο αριθμό 20. Αυτό δεν είναι απαραίτητα πρόβλημα (αν π.χ. θέλουμε στη συνέχεια απλά να εκτυπώσουμε την τιμή της `age`), αλλά εάν η τιμή της `age` πρόκειται να πάρει μέρος σε κάποια πράξη, τότε θα πρέπει να μετατραπεί σε `int`.

Για τη **μετατροπή τύπων** μεταβλητών, υπάρχουν οι εξής συναρτήσεις:

- Μετατροπή σε ακέραιο: `int()`
- Μετατροπή σε πραγματικό: `float()`
- Μετατροπή σε συμβολοσειρά: `str()`

Π.χ.,

```
age = input('Δώσε την ηλικία σου: ') # Η age είναι τύπου str
age = int(age)  # Μετατροπή της τιμής της age σε ακέραιο
                # και αποθήκευσή της εκ νέου στην age. Άρα, η age
                # είναι πλέον τύπου int.
```

ή, πιο σύντομα:

```
age = int(input('Δώσε την ηλικία σου: '))
```

(Ο δεύτερος αυτός σύντομος τρόπος είναι αυτός που χρησιμοποιείται συνήθως για είσοδο αριθμητικών τιμών).

➔ Οι συναρτήσεις `int()` και `float()` λειτουργούν μόνο εάν τα ορίσματά τους είναι αποδεκτές αριθμητικές τιμές, δηλαδή είναι συμβολοσειρές αριθμητικών τιμών του τύπου στον οποίο ζητάμε να μετατραπούν.

Π.χ., η ακόλουθη είσοδος (22.5) προκαλεί σφάλμα αφού η συμβολοσειρά '22.5' δε μπορεί να μετατραπεί σε `int`:

```
>>> age = int(input('Δώσε την ηλικία σου: '))
Δώσε την ηλικία σου: 22.5
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    age = int(input('Δώσε την ηλικία σου: '))
ValueError: invalid literal for int() with base 10: '22.5'
```

Μετατροπή `float` σε `int`:

```
float_value = 5.7
int_value = int(float_value)     # int_value = 5
                                 # Αποκοπή δεκαδικού μέρους
```

```
float_value = -3.8
int_value = int(float_value)    # int_value = -3
```

Μετατροπή `int` *σε* `float`*:*

```
int_value = 4
float_value = float(int_value)  # float_value = 4.0
```

Επανεκχώρηση τιμών *διαφορετικού τύπου* σε μεταβλητές

Στην Python, σε αντίθεση με τις περισσότερες γλώσσες προγραμματισμού, μπορεί σε κάποια μεταβλητή να επανεκχωρηθεί τιμή *διαφορετικού* τύπου από αυτόν που ήταν η μεταβλητή. Σε αυτή την περίπτωση, ο τύπος της μεταβλητής αλλάζει. Π.χ.,

```
>>> x = 50
>>> print(x)
50
>>> type(x)
<class 'int'>
>>> x = 'Hello!'
>>> print(x)
Hello!
>>> type(x)
<class 'str'>
```

Περισσότερα για την `print`

i) Εκτύπωση πολλαπλών ορισμάτων

Η συνάρτηση `print` μπορεί να δεχτεί πολλαπλά ορίσματα, τα οποία τα εκτυπώνει συνεχόμενα αφήνοντας μεταξύ τους έναν κενό χαρακτήρα. *Τα πολλαπλά ορίσματα μιας συνάρτησης χωρίζονται μεταξύ τους με κόμμα.* Μερικά παραδείγματα (με έντονα γράμματα η είσοδος από τον χρήστη):

```
>>> print('Δευτέρα','Τρίτη','Τετάρτη')
Δευτέρα Τρίτη Τετάρτη
>>> name = input("What's your name? ")
What's your name? George
>>> print('Hello',name)
Hello George
```

ii) Εκτύπωση χωρίς κενά με τον τελεστή `sep`

Εάν στο προηγούμενο παράδειγμα θέλαμε να εκτυπώσουμε το “Hello George” με π.χ. θαυμαστικό στο τέλος, θα είχαμε το πρόβλημα εισόδου ενός ανεπιθύμητου κενού:

```
>>> print('Hello',name,'!')
Hello George !
```

Στις περιπτώσεις που δεν επιθυμούμε την είσοδο κενών μεταξύ συμβολοσειρών ορισμάτων της `print`, χρησιμοποιούμε τον τελεστή `sep` ως τελευταίο όρισμα στη συνάρτηση, ως εξής:

```
print('συμβολοσειρά 1', 'συμβολοσειρά 2', sep = '')
```

Π.χ.,

```
>>> print(1, 2, 3)
1 2 3
>>> print(1, 2, 3, sep = '')
123
>>> print('Hello ', name, '!', sep = '')
Hello George!
```

Προσέξτε ότι στην τελευταία `print`, προσθέσαμε ένα κενό στο τέλος της λέξης Hello, γιατί διαφορετικά θα εμφανιζόταν ενωμένη με το George. Ο τελεστής `sep` λειτουργεί για όλα τα ορίσματα της συγκεκριμένης `print`. Επίσης, μπορεί να χρησιμοποιηθεί για να παρεμβάλει οποιοδήποτε σύνολο χαρακτήρων, π.χ.:

```
>>> print(1, 2, 3, sep = '***')
1***2***3
```

iii) Εκτύπωση χωρίς αλλαγή γραμμής με τον τελεστή `end`

Η συνάρτηση `print` εκτυπώνει στην οθόνη τα ορίσματά της και στη συνέχεια αλλάζει γραμμή. Για να αποτραπεί αυτό, ώστε η επόμενη κλήση της print να συνεχίσει να εκτυπώνει στην ίδια γραμμή, χρησιμοποιείται ο τελεστής `end` ως τελευταίο όρισμα στη συνάρτηση, ως εξής:

```
print('συμβολοσειρά', end = ' ')
```

Συνήθως στον τελεστή `end` εκχωρείται ο κενός χαρακτήρας, έτσι ώστε να υπάρχει ένα κενό μεταξύ της προηγούμενης και της επόμενης εκτύπωσης, όμως αυτό δεν είναι απαραίτητο και μπορεί να εκχωρηθεί απλά η κενή συμβολοσειρά:

```
print('συμβολοσειρά', end = '')
```

Π.χ., τα:

```
print('Ένα', end = ' ')
print('Δύο', end = ' ')
print('Τρία')
```

εκτυπώνουν: `Ένα Δύο Τρία`

ενώ τα:

```
print('Ένα', end = '')
print('Δύο', end = '')
print('Τρία')
```

εκτυπώνουν: `ΈναΔύοΤρία`

iv) Ακολουθίες διαφυγής

Οι ακολουθίες διαφυγής (escape characters) είναι συγκεκριμένοι συνδυασμοί χαρακτήρων με τον χαρακτήρα διαφυγής (\). Όταν χρησιμοποιείται μόνος του (όχι μέσα σε συμβολοσειρά), ο χαρακτήρας διαφυγής δίνει τη δυνατότητα συνέχισης μιας εντολής σε περισσότερες από μία γραμμές:

```
print('Η τιμή του συγκεκριμένου προϊόντος είναι', \
          price, 'ευρώ.')
```

(προσοχή στο ότι μία συμβολοσειρά δε μπορεί να χωριστεί σε πολλές γραμμές με αυτόν τον τρόπο)

Σε συνδυασμό με συγκεκριμένους χαρακτήρες, ο χαρακτήρας διαφυγής δημιουργεί τις ακόλουθες ακολουθίες διαφυγής (οι οποίες αποτελούν μέρος κάποιας συμβολοσειράς):

`\n` Αλλαγή γραμμής
`\t` Tab (επόμενος οριζόντιος στηλοθέτης)
`\'` Εμφάνιση αποστρόφου
`\"` Εμφάνιση διπλών εισαγωγικών
`\\` Εμφάνιση ανάστροφης καθέτου (\)

Π.χ.,

```
>>> print('1\n2\n3')
1
2
3
>>> print('Δευτ\tΤρ\tΤετ\nΠεμ\tΠαρ\tΣαβ')
Δευτ  Τρ   Τετ
Πεμ   Παρ  Σαβ
```

v) Ο τελεστής +

Ο τελεστής + μεταξύ δύο συμβολοσειρών πραγματοποιεί συνένωσή τους (string concatenation).

```
>>> my_text = 'Αυτό είναι ' + 'μία συμβολοσειρά'
>>> print(my_text)
Αυτό είναι μία συμβολοσειρά
```

Προσοχή χρειάζεται στο ότι η συνένωση γίνεται χωρίς την προσθήκη ενδιάμεσου κενού χαρακτήρα. Εάν κάτι τέτοιο είναι απαραίτητο, θα πρέπει να προβλεφθεί στο τέλος της πρώτης ή στην αρχή της δεύτερης συμβολοσειράς.

Άρα, πώς αλλιώς θα μπορούσε να γραφεί το παράδειγμα:

```
>>> print('Hello ', name, '!', sep = '')
Hello George!
```

χωρίς να είναι απαραίτητη η πρόβλεψη κενού μετά τη λέξη Hello;

Απάντηση:

```
>>> print('Hello', name + '!')
Hello George!
```

Δηλαδή μπορούμε να χρησιμοποιούμε ξεχωριστά ορίσματα στην `print` *εκεί που θέλουμε να υπάρχει κενό ανάμεσά τους, και τον τελεστή* `+` *εκεί που δεν θέλουμε να υπάρχει κενό (όταν πρόκειται αποκλειστικά για συμβολοσειρές και όχι για αριθμητικές μεταβλητές).*

Επίσης, προσοχή χρειάζεται στο ότι ο τελεστής λειτουργεί αποκλειστικά με συμβολοσειρές ή μεταβλητές τύπου `str`. Στην περίπτωση που χρησιμοποιηθεί μεταβλητή αριθμητικού τύπου (π.χ. `int`), προκύπτει σφάλμα, αφού ο διερμηνέας δεν κάνει αυτόματη μετατροπή του αριθμητικού τύπου σε `str`, όπως κάνει όταν βρίσκει μια τέτοια μεταβλητή ή τιμή ως όρισμα της `print`. Π.χ, το:

```
x = 5
print(x)
```

μετατρέπει την τιμή του x (όχι το x) σε συμβολοσειρά και την εκτυπώνει. Το ίδιο κάνει και το:

```
print(1)
```

που έχει ακριβώς το ίδιο αποτέλεσμα με το:

```
print('1')
```

Όμως, στην περίπτωση του τελεστή +, το:

```
age = 20
print('Η ηλικία σου είναι ' + age)
```

είναι λάθος, αφού η τιμή του `age` δε μετατρέπεται αυτόματα σε `str`, οπότε η αρχική συμβολοσειρά δε μπορεί να συνενωθεί με έναν ακέραιο. Για να γίνει αυτό, θα πρέπει να κάνουμε εμείς τη μετατροπή, με τη χρήση της συνάρτησης `str()`:

```
age = 20
print('Η ηλικία σου είναι ' + str(age))
```

Τέλος, όταν ο τελεστής + χρησιμοποιείται μεταξύ αριθμητικών τύπων μέσα σε ένα όρισμα της `print`, λειτουργεί κανονικά ως τελεστής πρόσθεσης, π.χ.:

```
>>> x = 1
>>> y = 2
>>> print('Το άθροισμα είναι', x + y)
Το άθροισμα είναι 3
```

vi) Ο τελεστής `*`

Ο τελεστής `*` μεταξύ μιας συμβολοσειράς και ενός ακεραίου, επαναλαμβάνει τόσες φορές τη συμβολοσειρά όσες ορίζει ο ακέραιος. Π.χ.,

```
>>> print('a' * 10)
aaaaaaaaaa
>>> print(5*'*' + 3*'.')
*****...
```

vii) Μορφοποίηση αριθμών

Υπάρχει η ενσωματωμένη συνάρτηση `format` η οποία μορφοποιεί την εμφάνιση αριθμητικών τιμών. Καλείται ως εξής:

```
format(αριθμητική_τιμή, τελεστής_προσαρμογής)
```

και μορφοποιεί την αριθμητική τιμή που δέχεται ως πρώτο όρισμα, σύμφωνα με τον τελεστή προσαρμογής που δέχεται ως δεύτερο όρισμα. Η μορφοποίηση είναι συνήθως απαραίτητη στην εμφάνιση πραγματικών αριθμών. Π.χ.,

```
>>> num = 34567
>>> den = 12
>>> fraction = num/den
>>> print(fraction)
2880.5833333333335
```

Υπάρχουν οι εξής δυνατότητες μορφοποίησης:

α) Πλήθος δεκαδικών ψηφίων: `format(number, '.xf')`

όπου `x` είναι το πλήθος των δεκαδικών ψηφίων του αριθμού `number` που θα εμφανισθούν. Π.χ.,

```
>>> print(format(fraction, '.3f'))
2880.583
>>> print(format(fraction, '.1f'))
2880.6
```

Ο αριθμός *στρογγυλοποιείται* στο συγκεκριμένο πλήθος δεκαδικών ψηφίων.

β) Καθορισμός εύρους πεδίου: `format(number, 'y.xf')`

όπου `x` είναι ο αριθμός των δεκαδικών ψηφίων του `number` που θα εμφανισθούν και `y` είναι ο ελάχιστος αριθμός θέσεων που θα χρησιμοποιηθούν για την εμφάνιση του αριθμού (εύρος πεδίου). Π.χ.,

```
>>> print('Το αποτέλεσμα είναι', format(fraction, '12.3f'))
Το αποτέλεσμα είναι     2880.583
```

γ) Εισαγωγή διαχωριστικών κομμάτων: `format(number, ',.xf')`

όπου `x` είναι ο αριθμός των δεκαδικών ψηφίων του `number`, ο οποίος θα εμφανισθεί με διαχωριστικά κόμματα στις χιλιάδες (το `.x` είναι προαιρετικό):

```
>>> print('Το αποτέλεσμα είναι', format(fraction, ',.3f'))
Το αποτέλεσμα είναι 2,880.583
```

δ) Εκθετική μορφή: `format(number, '.xe')`

όπου `x` είναι ο αριθμός των δεκαδικών ψηφίων του `number`, ο οποίος θα εμφανισθεί σε εκθετική μορφή (το `.x` είναι προαιρετικό):

```
>>> print('Το αποτέλεσμα είναι', format(fraction, 'e'))
Το αποτέλεσμα είναι 2.880583e+03
```

ε) Μορφή ποσοστού: `format(number, '.x%')`

όπου `x` είναι ο αριθμός των δεκαδικών ψηφίων του `number`, ο οποίος θα εμφανισθεί σε μορφή ποσοστού % (το `.x` είναι προαιρετικό):

```
>>> print(format(0.472385, '.2%'))
47.24%
```

στ) Μορφοποίηση ακεραίων: Με τον τελεστή `d` αντί για `f`.

Ουσιαστικά η μορφοποίηση ακεραίων αφορά τη δυνατότητα καθορισμού εύρους πεδίου και εισαγωγής διαχωριστικών κομμάτων χιλιάδων. Π.χ.,

```
>>> print(format(123456, ',d'))
123,456
>>> print(format(123456, '12d'))
      123456
```

Εκτέλεση υπολογισμών

Μαθηματικοί τελεστές της Python:

Τελεστής	Πράξη
+	Πρόσθεση
-	Αφαίρεση
*	Πολλαπλασιασμός
/	Διαίρεση
//	Ακέραια διαίρεση
%	Υπόλοιπο διαίρεσης
**	Ύψωση σε δύναμη

Παραδείγματα:

```
>>> 15 + 2.5
17.5
>>> 5/2
2.5
>>> 5//2
2
>>> -5//2
-3
>>> 2**5
32
>>> 11%3
2
```

(Η ακέραια διαίρεση κάνει στρογγυλοποίηση προς το μείον άπειρο)

Ένα απλό παράδειγμα προγράμματος:

```
# Το πρόγραμμα αυτό διαβάζει την αρχική τιμή ενός προϊόντος
# και υπολογίζει τη μειωμένη τιμή του, με έκπτωση 20%.

# Είσοδος της αρχικής τιμής του προϊόντος.
original_price = float(input('''Δώσε την αρχική τιμή
                                 του προϊόντος: ''')) 

# Υπολογισμός του ποσού της έκπτωσης.
discount = original_price * 0.2

# Υπολογισμός της μειωμένης τιμής.
sale_price = original_price - discount

# Εμφάνιση της τελικής τιμής πώλησης.
print('Η τελική τιμή πώλησης είναι', sale_price)
```

<u>Προτεραιότητα τελεστών και υπολογισμών:</u>

Οι μαθηματικοί τελεστές έχουν την εξής προτεραιότητα εκτέλεσης:

1. Ύψωση σε δύναμη (`**`)
2. Πολλαπλασιασμός, διαίρεση και υπόλοιπο διαίρεσης (`*`, `/`, `//`, `%`)
3. Πρόσθεση και αφαίρεση (`+`, `-`)

Πράξεις ίδιου βαθμού προτεραιότητας εκτελούνται από αριστερά προς τα δεξιά, με εξαίρεση την ύψωση σε δύναμη, όπου πολλαπλές υψώσεις εκτελούνται από δεξιά προς τα αριστερά.

Π.χ.,

```
>>> result = 15 + 5 / 2
>>> print(result)
17.5
```

(Το αποτέλεσμα δεν είναι 10 γιατί πρώτα πραγματοποιείται η διαίρεση και μετά η πρόσθεση)

```
>>> result = 5 + 12 / 3 * 2
>>> print(result)
13.0
```

(Πρώτα εκτελείται η διαίρεση και μετά ο πολλαπλασιασμός)

```
>>> result = 4**3**2
>>> print(result)
262144
```

(Το 262144 είναι το 4^9, δηλαδή πρώτα εκτελείται το 3^2 και μετά η αρχική ύψωση)

Ο τρόπος παρέμβασης στην προτεραιότητα των πράξεων από τον προγραμματιστή γίνεται με τη χρήση παρενθέσεων. Π.χ.,

```
>>> result = (15 + 5) / 2
>>> print(result)
10.0
```

(Προσέξτε ότι το αποτέλεσμα της κανονικής διαίρεσης είναι πάντα τύπου float, ακόμα και αν το αποτέλεσμα είναι ακέραιος αριθμός)

Άρα, συνολικά οι μαθηματικοί υπολογισμοί εκτελούνται ως εξής:

1. Παρενθέσεις
2. Πράξεις ανάλογα με την προτεραιότητα των τελεστών
3. Σε περίπτωση ίδιας προτεραιότητας τελεστών: από αριστερά προς τα δεξιά (η ύψωση σε δύναμη, ανάποδα)

Άλλα παραδείγματα:

- Το κλάσμα: $\frac{a \cdot b}{c+d}$: `a * b / (c + d)` και *όχι*: `a * b / c + d`

  Ο πλεονασμός στις παρενθέσεις *δεν* είναι λάθος. Π.χ., το: `(a * b) / (c + d)` είναι και αυτό σωστό, παρόλο που οι παρενθέσεις στο `(a * b)` δεν είναι απαραίτητες.

- Ο μέσος όρος των τιμών π.χ. τριών μεταβλητών `a`, `b` και `c`:
  `(a + b + c) / 3` και όχι: `a + b + c / 3`

Παράδειγμα: Πρόγραμμα μετατροπής δευτερολέπτων σε ώρες/λεπτά/δευτερόλεπτα.

```
# Είσοδος του πλήθους δευτερολέπτων από τον χρήστη.
total_seconds = float(input('Δώσε πλήθος δευτερολέπτων: '))
# Υπολογισμός του πλήθους των ωρών.
hours = total_seconds // 3600
# Υπολογισμός του πλήθους των λεπτών που απομένουν.
minutes = (total_seconds // 60) % 60
# Υπολογισμός του πλήθους των δευτερολέπτων που απομένουν.
seconds = total_seconds % 60
# Εμφάνιση των αποτελεσμάτων.
print('Ο χρόνος σε ώρες, λεπτά και δευτερόλεπτα:')
print('Ώρες:', hours)
print('Λεπτά:', minutes)
print('Δευτερόλεπτα:', seconds)
```

Παράδειγμα: Πρόγραμμα μετατροπής μέσου χρόνου δρομέα από "ανά χιλιόμετρο" σε "ανά μίλι".

```
# Είσοδος μέσου χρόνου το χιλιόμετρο (πρώτα τα λεπτά, μετά τα δευτερόλεπτα)
min_km = int(input('Μέσος χρόνος το km - Λεπτά: '))
sec_km = int(input('Μέσος χρόνος το km - Δευτερόλεπτα: '))
total_sec_km = min_km * 60 + sec_km  # Συνολικός χρόνος σε δευτερόλεπτα
total_sec_mi = 1.60934*total_sec_km  # Συνολικός χρόνος (sec) για το μίλι
min_mi = total_sec_mi // 60   # Το πλήθος των λεπτών
sec_mi = total_sec_mi % 60    # Τα δευτερόλεπτα που απομένουν

print('Ο χρόνος', str(min_km) + ':' + str(sec_km), 'το χιλιόμετρο')
print('ισοδυναμεί με', str(format(min_mi, '.0f')) + ':' \
              + str(format(sec_mi, '.0f')), 'το μίλι.')
# Ή αλλιώς:
# print('ισοδυναμεί με', str(int(min_mi,)) + ':' \
#                 + str(int(sec_mi)), 'το μίλι.')
```

Παράδειγμα εξόδου (με έντονους χαρακτήρες η είσοδος):

```
Μέσος χρόνος το km - Λεπτά: 5
Μέσος χρόνος το km - Δευτερόλεπτα: 25
Ο χρόνος 5:25 το χιλιόμετρο
ισοδυναμεί με 8:43 το μίλι.
```

Τελεστές επαυξημένης εκχώρησης

Κάποιες πράξεις και εκχωρήσεις τιμών που περιλαμβάνουν εκχώρηση του αποτελέσματος σε μεταβλητή που παίρνει μέρος στην πράξη, μπορούν να συντομευθούν με τη χρήση των τελεστών επαυξημένης εκχώρησης (+=, -=, *=, /=. %=), ως εξής:

```
a += x;   ↔  a = a + x;
a -= x;   ↔  a = a - x;
a *= x;   ↔  a = a * x;
a /= x;   ↔  a = a / x;
a %= x;   ↔  a = a % x;
```